Low-Latency Reconfigurable Entropy Digital True Random Number Generator With Bias Detection and Correction

Leonardo Bosco Carreira, *Student Member, IEEE*, Paige Danielson, *Student Member, IEEE*,
Arya A. Rahimi, *Student Member, IEEE*, Maximiliam Luppe, *Member, IEEE*,
and Subhanshu Gupta, *Senior Member, IEEE*

***Abstract*—Digital true-random number generators (TRNG) are increasingly employed to generate random channels in low-power resource-constrained IoT devices at the network edge. However, their susceptibility to process variations, or even intrusion attacks, degrade the generated entropy requiring an on-the-fly processor for detection of bias variations and correction. This work proposes a two-step search process to implement an optimized search that minimizes the latency (number of clock-cycles) for bias correction implemented on a FPGA platform. The first step implements a subset of NIST tests for entropy validation and an additional autocorrelator is used for entropy validation and bias detection on-the-fly in the second step. Measured results with the proposed algorithm implemented on FPGA shows significant improvement in the probability of bias correction with low number of trials. The measured power consumption of the TRNG and the bias correction is 10.22mW and 10.96mW respectively at 1.25 V with 18 kHz throughput for three random channels.**

***Index Terms*—Digital true-random number generator, reconfigurable, bias detection and correction, low-latency.**

I. Introduction

THE dramatic growth of Internet-of-Things (IoT) [1], [2] based applications have raised new concerns about robust security for resource-constrained devices at the network edge. These devices demand on-sensor decision making, autonomous bias detection and key regeneration without the overhead of cloud-driven secure exchange [3].

Although various threats challenge the security of IoT, the root of trust starts from the hardware security [2], [4]. Fig. 1 shows a typical edge-device (for example, an IoT gateway or a router at the network edge) that has a random-number generator (RNG) attached to its modules through high-speed interconnect. High entropy RNGs mitigate the threat of revealing sensitive information from a system. For this reason, different sources of randomness have been used in the past including cryptographically-secured pseudo random number generators (PRNG) [3], analog or digital true-random number generators (TRNG) [5]–[7]. The periodicity of PRNGs with a fixed pattern results in spurs and requires long bit sequence generators that can constrain the system power budget. TRNGs in contrast, harvest entropy from physical sources without any periodicity. Furthermore, even if high quality PRNGs may be built, these PRNGs still need to be seeded using TRNGs [8].

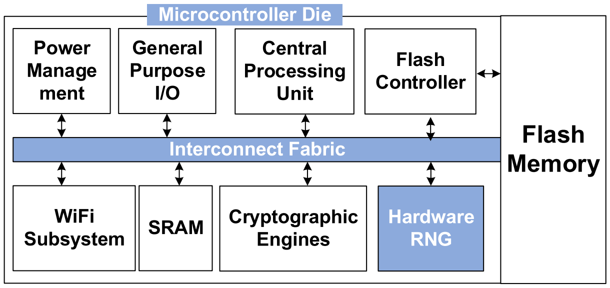


Fig. 1. Conventional microcontroller die with hardware random number generator (Hardware RNG).

To date, various combinations of analog and digital components have been proposed [8]–[13] as sources of random jitter. From a practical standpoint, it is highly desirable to construct the TRNGs using digital design techniques through cheap bulk-silicon processes. Jitter and metastability are two main sources of randomness in digital TRNGs with ring-oscillator (RO) based TRNG designs gaining popularity owing to their simplicity and portability across technology nodes. The operation mechanism of the RO-based digital TRNG can be classified into free-running [8], [11], [12] and staged-running mechanism [14] with emphasis on higher entropy outputs.

While simple to design, ROs are susceptible to PVT variations (fabrication process, supply voltages and operating temperature). If no compensation is made to combat these issues, it can result in highly variable and unreliable entropy generation between different fabricated parts. This unreliability results in a statistical imbalance in the numbers of ‘1’s and ‘0’s termed as bias imbalance. The device characteristics and the varying PVT environment makes it harder to predict this bias before implementation. Efforts to increase the entropy

Manuscript received April 15, 2019; revised October 3, 2019 and November 26, 2019; accepted December 10, 2019. Date of publication January 8, 2020; date of current version May 1, 2020. This work was supported in part by the Washington State University (WSU) Grand Challenges Research Grant Award. The work of S. Gupta was supported by the Cisco Faculty Research Award #935126. This article was recommended by Associate Editor R. Azarderakhsh. *(Corresponding author: Subhanshu Gupta.)*

L. B. Carreira, P. Danielson, A. A. Rahimi, and S. Gupta are with the School of EECS, Washington State University, Pullman, WA 99163 USA (e-mail: sgupta@eecs.wsu.edu).

M. Luppe is with the Department of Electrical Engineering, University of São Paulo, São Paulo 01000, Brazil.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2019.2960694

of RO-based TRNGs alleviating issues from bias variation have included combining outputs of several parallel ROs [8], dynamic duty-cycle tuning [15] and more recently, collapsible even-stage RO with automatic tuning loop [16]. In practice, the raw random bitstream from the entropy generator does not bear satisfactory statistical properties, therefore, the auxiliary post-processing part performs the tasks of quality improvement. However, the dependence on post-processing brings in throughput loss and potential security weakness [17].

This work proposes an embedded host-processor algorithm (HP) that detects and corrects bias variations in the digital TRNG on-the-fly achieving up to 98% success in recovery. This is achieved by combining two methods creating a novel recovery algorithm implemented on hardware: A learning-based method using a subtest of NIST tests [18] to store the RO paths with higher entropy and the AIS-20/31 [19] autocorrelation test to quickly detect output bias and select the stored paths until the output is unbiased. Both combined were able to greatly reduce the recovery latency. The main contributions of this work are as follows:

a. We propose two metrics to select a subset from NIST tests for lightweight hardware implementation. The selected subset is uncorrelated and sensitive to output bias. (Section III)
b. We introduce a learning-based method based on hardware implementation of NIST tests that greatly reduces the number of cycles needed to detect and correct output bias compared to the state-of-the-art. (Section IV)
c. We demonstrate a complete hardware implementation with a dedicated host processor for bias detection and correction based on lightweight NIST/AIS test implementation (Section V).
d. Finally, the randomness property of the proposed TRNG is validated using all NIST and AIS-20/31 statistical randomness tests. (Section VI)

The proposed algorithm is divided in two different stages called the *learning mode* and the *running mode*. During the learning mode, the configuration bits that generates higher entropy outputs are saved in FPGA memory. This selection is made based on a subset of NIST tests implemented on the FGPA and applied to the TRNG output generated by each configuration bits' combination. The *learning mode* is completed with storage configurations learned in a controlled environment. The system then executes the *running mode* where it uses an autocorrelator to identify output bias on-the-fly. When bias variation is detected, system recovery is initiated driven by the guided search that uses the configuration settings stored in the memory during learning mode to produce unbiased outputs again. The TRNG is implemented on a development kit equipped with a Cyclone IV FPGA with the HP integrated on a Cyclone V FPGA.

Section II briefly discusses the RO-based digital TRNG. Section III defines the randomness tests for entropy determination on hardware followed by the proposed algorithm in Section IV. Section V presents the hardware implementation and section VI presents the measured results under bias variations. Lastly, Section VII concludes this paper with potential research areas in future.

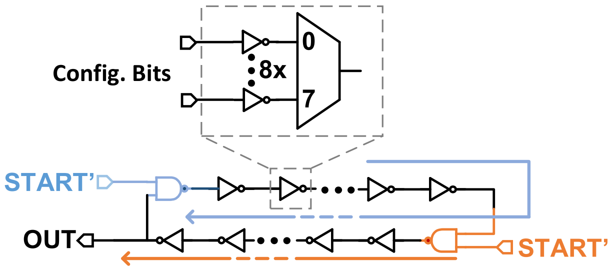


Fig. 2. RO proposed by Yang *et al.* [16].

II. Prior Art

A. RO-Based Entropy Generator

The manifestation of random jitter in digital RNGs was discussed earlier in the seminal work by Abidi [20]. Fluctuations in zero-crossing instants of the inverter output ramps were modeled as random jitter and then linked to phase noise. Yang *et al.* [16] extended the model in [20] using collapse time of a dual-edge inverter-based RO (shown in Fig. 2) as a randomness source. The same signal (START) is injected into the ring by two NAND gates. However, the independent random noise effects associated with different rising and falling times from the propagation delay of the two injected edges cause the oscillation to stop after a finite number of cycles due to the collapse of the falling and rising edge. A counter counts the number of pulses before this collapse occurs. The output of this counter is then stored as the random number output. The entropy of this TRNG was monitored using an off-chip host-processor unit that analyses the number of cycles taken to collapse multiple times and alters the configuration of the RO when any shift in the mean of the collapse times is detected. To ensure high entropy in presence of PVT variations, each stage of the RO was replaced by eight selectable inverters controlled by three input multiplexers, the selection bits of each stage is called configuration bits. This selection makes it possible to rearrange the oscillation paths accounting for any bias variations. Thus, for S inverter stages in the RO, 8^S possible configurations are created with each configuration exhibiting a slightly different oscillation frequency. The above architecture proved that the digital RO based TRNG can achieve high entropies while generating multiple random channels, being each counter output considered as a channel. However, to correct the bias variations the TRNG must search for a new configuration among all 8^S until the output became unbiased what requires multiple trials, this searching process is named in this work as random search. Every time that the TRNG in [16] detected a biased output and must change its configuration bits it took anywhere between 12.5 configurations (considering 8% success rate in typical condition) to 315 configurations (worst-case) to correct for the bias variations. Each configuration further requires anywhere between 500 to 5000 trials (collapses) as the reported lower and upper bounds to calculate the mean of cycles taken to collapse used as correction metric. Thus, the total trials required can vary between 6250 (= 12.5 × 500) in the usual case to 1,575,000 (= 315 × 5000) in the worst case. In [16], the random search approach becomes highly computationally intensive. The bias detection and correction methodology were further implemented off-chip only. This work overcame the drawback in [16] by reducing the sample space of the number of configurations to be searched to a subset of pretested configurations which have higher probability of success.

B. Validation of Random Number Generators

Several sophisticated tests have been proposed [18], [19], [21] and metrics developed to analyze and validate various aspects of randomness. The National Institute of Standards and Technologies (NIST) RNG test suite [18] and the AIS-20/31 [19] test suite provided by the German Federal Office for Information in Security's (BSI) are the most popular used to RNG validation, both are an extension of the Federal Information Processing Standards (FIPS) tests [21]. However, hardware implementation of these tests is too computationally intensive. Over the years lightweight hardware implementations have been studied. Deciding which test to implement among all existing is critical to reduce the hardware complexity. Moreover, implementing all the tests is not viable due to increasingly area and power constraint designs [22]. The most common selection criteria is the input data size required to perform each test [22], [24] because the size of the hardware needed to compute them depends on the required data sample size. The tests also must be simplified to reduce the computation time. In [22], [23], and [24], a similar method with relaxed computational effort was proposed by reducing each test to simple logic modules. However, no closed-loop bias correction was proposed. As proposed in Section III, this work will extend these prior studies by selecting a bias sensitive and uncorrelated subset of the tests that do not add additional power and area overhead. The selection of tests for bias detection is further inspired by [24] where it is shown that the first-time lag in the AIS-20/31 autocorrelation tests is a very sensitive parameter that can be used for bias detection. The proposed method goes beyond closing the loop by automatically controlling the entropy source for bias correction after bias detection.

III. Tests Definition For Entropy Determination

This section defines the various tests used in entropy determination in the proposed algorithm described in Section IV. It is important to note that our proposed approach is completely digital and does not have any analog components. We use the NIST test suite [18] with an additional autocorrelator for entropy validation. Though an exhaustive validation using all the 16 tests in the NIST test suited can be conducted offline, we use a subset of NIST tests in this work to save hardware resources without loss in accuracy. We also confirm the entropy generation results by reading the random bits offline and conducting all NIST tests for validation as described in Section V.

In order to determine the most suitable subset of NIST tests, we used the following selection criteria: i) effort required for hardware computation, ii) sensitivity to bias variations, and iii) degree of correlation. Tests with lower hardware effort (i.e. the smaller input size and the number of gates requirements) had priority over those that demand higher effort. The number of bits necessary to perform each test were based on the NIST test suite documentation [18]. The sensitivity of these tests was analyzed using the sum-of-squared-error (SSD) of NIST tests results for biased and unbiased sequences. Following this, each of the NIST tests are analyzed pairwise, and their degree of correlation is observed to exclude redundant pairs. The selection process is described in Fig. 3 and Fig. 4 respectively.

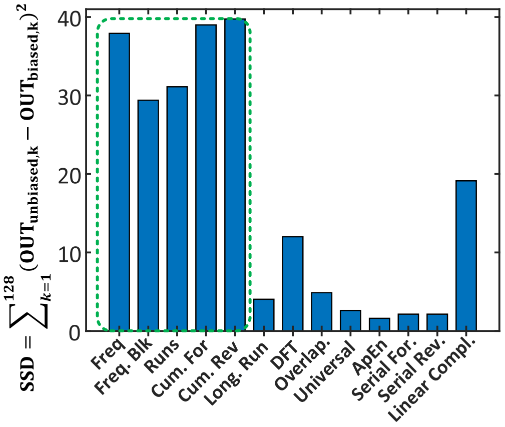


Fig. 3. Sum of Squared Error (SSD) of NIST test results of biased and unbiased sequences. The Non-Overlapping, Random Excursion, and Random Excursion Variant tests were not chosen due to large computational overhead.

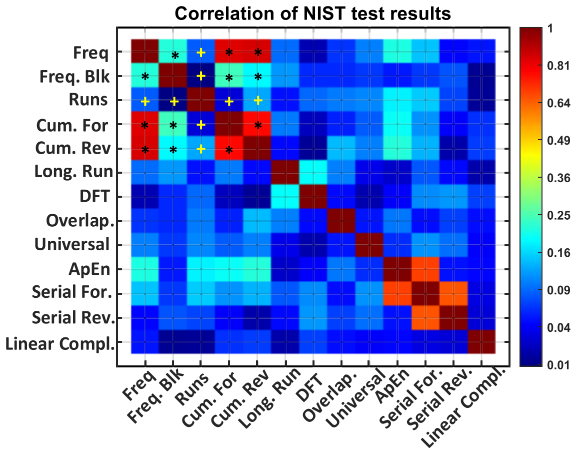


Fig. 4. Correlation matrix of NIST tests results plotted for a random input sequence. ∗(black) High degree of correlation. +(yellow) Low degree of correlation.

Fig. 3 analyzes the sensitivity of each test to input bias after determining the number of bits based on NIST test suite [18]. A random input vector of 128 × 10^6 bits created using MATLAB RNG was split in 128 unbiased sequences of 10^6 bits, and used as an input for the NIST tests suite. Each test returned 128 unbiased outputs ($OUT_{unbiased}$). Later, each sequence with 10^6 bits were purposely biased injecting '1's at random positions at the limit where the first test starts to fail. Using the obtained sequences as new inputs to the test suite, 128 new results (OUT_{biased}) were obtained for each test. The computed SSD between the biased and the unbiased test outputs is shown in Fig. 3.

Fig. 3 shows that the Longest Run of Ones (Long. Run) and the DFT tests (DFT) are not as sensitive to bias change as

Frequency (Freq), Frequency within Block (Freq. Blk), Runs (Runs), Cumulative Sum Forward (Cum. For) and Cumulative Sum Reverse (Cum. Rev). The unbiased results were then used to calculate the correlation between each test as shown in the correlation matrix in Fig. 4. It can be observed that the Runs test exhibits a very low degree of correlation with all other tests as well as has low computational complexity. This analysis resulted in three pairs of tests that meet the desired expectations: 1) Runs and Frequency Monobit, 2) Runs and Cumulative Sum (Forward or Reverse), and 3) Runs and Frequency within a Block. The Frequency within a Block test implementation consumes 40% more area and power than the Frequency Monobit test [22]. The Cumulative Sum (Forward or Reverse) and the Frequency Monobit tests in pairs 1 and 2 calculate the same statistical variable (partial sum) which makes them highly correlated as shown in Fig. 3. As either of the pairs 1 and 2 could be chosen, the Runs and Frequency Monobit test in pair 1 has been selected because both are requirements of other important standard FIPS 140-2 [21] and AIS-20/31 [19].

These subtests including the Frequency Monobit and the Runs tests guarantee the higher channel entropy configurations to be selected and stored during the training process. We describe these tests as follows:

1. Frequency Monobit test: This test is the preliminary test of randomness and all other tests depends on passing of this test [18]. This test calculates the ratio of zeros and ones in the sequence under analysis and compares with the desired scenario where the number of zeros and ones should be equal. The input sequence bits, ϵ_{i}, are summed following the function below [18]:

$$S_{\mathrm{u}} = \sum_{i=1}^{u} 2 \times \epsilon_{\mathrm{i}} - 1 \tag{1}$$

where u is the number of bits of the sequence under analysis. We further determine the range of acceptable proportions using the confidence interval ($\alpha = 0.01$) as:

$$p' \pm 3\sqrt{\frac{p'(1-p')}{m}} \tag{2}$$

where, $p' = 1 - \alpha$ and m is the number of sequences with u bits. For $m = 128$, greater than the NIST recommendation for $\alpha = 0.01$ [18], this requires 96% of sequences to have $p_{\mathrm{value}} > 0.01$. In other words, 123 out of 128 sequences must have their $p_{\mathrm{value}} > 0.01$. In our implementation, u is selected to be 256, to have a larger margin from the minimum recommended value of 100 for this test [18]. The generated S_{u} is converted into a p_{value} using the complementary error function (*erfc*) [25] as shown in (3) by the following equation [19]:

$$\mathrm{P-value} = erfc(\frac{|S_{\mathrm{u}}|}{\sqrt{2 \times u}}) \tag{3}$$

The hardware implementation of *erfc*function, however, is computationally intensive. To overcome this problem, (3) was solved for S_{u}. Considering NIST recommendation of $\mathrm{P-value} \geq 0.01$, S_{n} was found to be:

$$-41 < S_{\mathrm{u}} < 41$$

This simplification reduces the hardware implementation of the Frequency Monobit test to a simple digital circuitry comprising of an accumulator and a binary comparator as described in Section V.

2. Runs test: This test is to quantify the number of uninterrupted equal bits sequences. While the Frequency Monobit test determines the ratio of zeros and ones, the Runs test determine if the sequence is oscillating at a slower or faster rate. Both these tests give independent answers and assess completely different aspects of randomness and hence are a good complement. Other combinations, such as the cusum test and the Frequency Monobit test, result in *P-value* that are likely to be correlated, as shown in Fig. 4. The test is divided in two stages. Given the sequence under test $\epsilon_{1:u}$, with length u, it calculates the number of ones in the sequence, k, as follows:

$$k = \sum_{j=1}^{u} \epsilon_{\mathrm{j}} \tag{4}$$

Using the value of k, the proportion of ones, π, in the sequence is then calculated as:

$$\pi = \frac{k}{u} \tag{5}$$

This leads to the calculation of the number of observed Runs, υ_{obs}, in the analyzed sequence as follows:

$$\upsilon_{\mathrm{obs}} = \sum\nolimits_{\mathrm{j}=1}^{u-1} r\,(\mathrm{j}) + 1 \tag{6}$$

where $r\,(\mathrm{j}) = 0$ if $\epsilon_{\mathrm{j}} = \epsilon_{\mathrm{j+1}}$, and $r\,(\mathrm{j}) = 1$ otherwise. Using (34)-(45), the computed *P-value* is defined as [18]:

$$\mathrm{P-value} = erfc(\frac{|\upsilon_{\mathrm{obs}} - 2u\pi\,(1-\pi)|}{2\sqrt{2u}\pi\,(1-\pi)}) \tag{7}$$

Similar to the Frequency Monobit test, the hardware implementation of *erfc* is computationally intensive. For the sequence to be considered satisfactory, the above equation is solved for two variables, k and υ_{obs} targeting P-value $\geq$ 0.01 [18]:

$$erfc\left(\frac{\left|\upsilon_{\mathrm{obs}} - 2k\left(1-\frac{k}{u}\right)\right|}{\frac{2\sqrt{2u}k}{u}\left(1-\frac{k}{u}\right)}\right) \geq 0.01 \tag{8}$$

As we are analyzing 128 sequences of 256 bits each, u is fixed at 256. Using a symbolic solver, k was varied for all possible sums of '1's in a 256 bits sequence, i.e. from 0 to 256. Because equation (8) is a quadratic equation, there are two solutions for each k. Solving this, we get a bounded $k \times 2$ matrix. Each υ_{obs} will be stored between lower and upper bounds labeled as $\upsilon_{\mathrm{l}}(k)$ and $\upsilon_{\mathrm{h}}(k)$, where k lies between 0 to 256, this process is later depicted in Fig. 6(a). As described in section IV, this solution reduces the hardware computational time requiring only a comparator to check if υ_{obs} is in the desired range for the calculated number of ones (k) in the sequence under analysis.

3. Max average collapse check: Large PVT variations can result in wrong estimate of collapse time as the RO may not collapse, and keeps oscillating. This check guarantees that the HP doesn't store a configuration that was oscillating close to

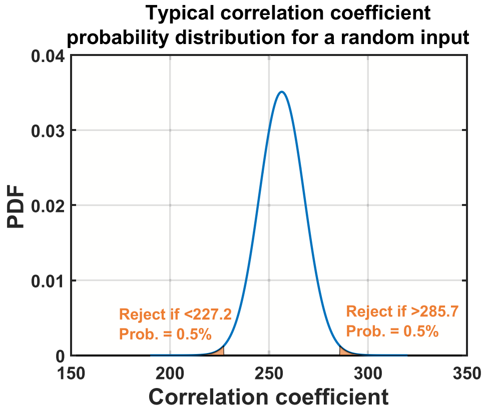


Fig. 5. Typical correlation coefficient distribution for a 512-bit random sequence [23].

the maximum allowed. As described in section V, the temperature increases after prolong usage can result in the number of cycles taken to collapse to increase by 25%. The mean and standard deviation of the stored configurations thus needs to accommodate this temperature increase. The Max Average Collapse Check thus ensures that recorded configurations with collapse times close to the maximum are not recorded.

4. Correlation coefficient: We calculate the auto-correlation coefficient as a measure of independence between two random variables. The first-time lag auto-correlation coefficient is defined by [19] and implemented in [24]:

$$C_1 = \sum_{\mathrm{k}=1}^{\mathrm{u}-\mathrm{i}} \epsilon_{\mathrm{k}} \oplus \epsilon_{\mathrm{k}+1} \tag{9}$$

where $a\epsilon_{\mathrm{k}}$ and $\epsilon_{\mathrm{k}+1}$ are the bits of the tested sequence. The upper and lower limits of the correlation coefficient can be fixed empirically from the mean and standard deviation of the probability density function (pdf). Considering a rejection of 1%, we can determine the upper and lower limits to accept or decline the sequence under analysis based on this statistical feature. As shown in Fig. 5, proven random sequences were generated in MATLAB and the rejection limits were calculated for 512-bit sequences on a total of 10×10^6 bits. The calculated correlation coefficient yields the same pdf and is also useful for bias detection in two different topologies of TRNG [24].

5. Power supply variations and its effect on entropy generation: We analyze the effect of power-supply variations to demonstrate change in bias followed by detection and correction. As the variance (σ^2) is related to the supply of a RO, any changes in the supply voltage induces a change in the variance as described in [20] and re-stated below:

$$\sigma^2 = \frac{4kT\gamma\, t_{\mathrm{dN}}}{I\,(V_{\mathrm{DD}} - V_{\mathrm{t}})} + \frac{kTC}{I^2} \tag{10}$$

Here, t_{dN} is the window that noise is integrated during output transition, I is the charging/discharging current for each inverter stage, V_{DD} is the supply voltage, V_{t} is the threshold voltage, γ is the technology-dependent noise coefficient, C is the load capacitance of the inverter, and k is the Boltzmann constant. Supply variation results in change in variance of the TRNG that requires a closed-loop feedback to ensure high degree of randomness in the entropy generation. The next section describes the proposed algorithm for low-latency (or few clock cycles) bias detection and correction.

TABLE I
Hardware Tests for Bias Variation Detection

Test	Learning mode	Running mode
Frequency Monobit test	X	
Runs test	X	
Max average collapse test	X	
Correlation coefficient		X

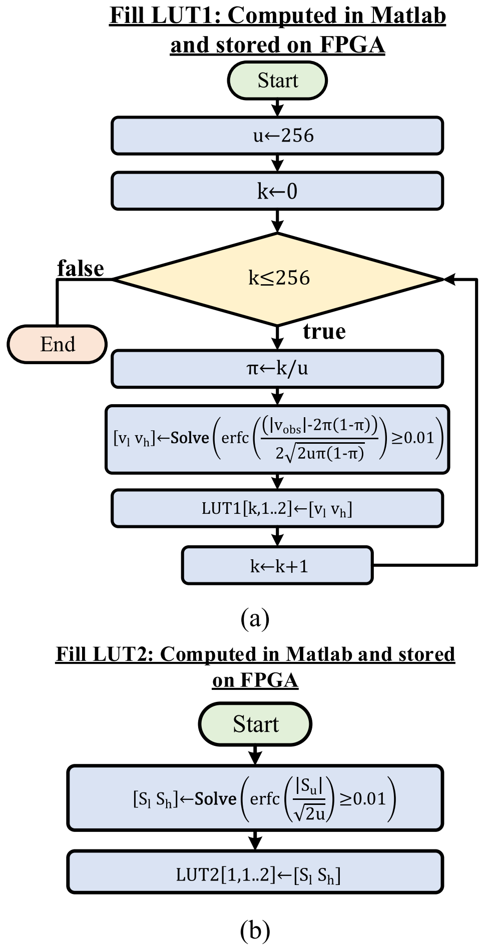
IV. Proposed Reconfigurable Entropy Generator with Bias Detection and Correction Algorithm

This section describes the proposed algorithm for the reconfigurable entropy generator with low-latency bias detection and correction. The proposed algorithm is separated into two stages both implemented in hardware. The first-stage implements a *learning mode* that identifies and stores the best ranked configurations exhibiting the highest entropy. The classification is based on a pipeline hardware implementation of a subset of NIST's randomness tests [18]. The data analysis is separated in two phases. The first phase executes the tests for every sequence of 256 collapses. The second phase process the results of all the 128 sequences. The three least significative bits (LSBs) of the collapses are analysed in parallel and must be approved for the configuration under test to be stored. The second-stage called as *running mode* estimates the entropy on-the-fly and uses the stored configurations to re-configure the RO when bias variations are detected.

Table I illustrates the tests executed in the *learning mode* comprising of the Frequency Monobit test, Runs test and Max Average Collapse Check as previously defined in Section III. It is important to note that the learning mode is executed only during initial configuration of the FPGA to identify and store the best ranked configurations. This mode is performed under a controlled environment to ensure that the output generated by each configuration bits tested reflects exactly the quality of the RO oscillation paths under tests, avoiding any possible bias caused by external sources. The learning and the running mode algorithm are described in detail next.

A. Learning Mode

To speed the learning process, the *erfc* calculations in (3) and (8) for Frequency Monobit and Runs tests, respectively are computed in MATLAB. The desired values of observed runs (v_{obs}) and partial sum (S_{u}) for the given sequence length (u) and number of ones in the sequence under test (k) are computed and thus stored in two mapped memories on the FPGA labeled as LUT1 (Fig. 6(a)) and LUT2 (Fig. 6(b)). LUT2 stores the minimum and maximum partial sum acceptable to guarantee P-value > 0.01 in Frequency Monobit test for a sequence of 256 bits. LUT1 stores the minimum and maximum observed runs acceptable for each case with number of ones (k) in the 256-bit sequence that guarantee *P-value* > 0.01 in Runs test.


Fill LUT1: Computed in Matlab and stored on FPGA
Start
u←256
k←0
k≤256
false
End
true
π←k/u
[v_l v_h]←Solve(erfc((|v_obs|-2π(1-π))/(2√(2uπ(1-π))))≥0.01)
LUT1[k,1..2]←[v_l v_h]
k←k+1
Fill LUT2: Computed in Matlab and stored on FPGA
Start
[S_l S_h]←Solve(erfc(|S_u|/√(2u))≥0.01)
LUT2[1,1..2]←[S_l S_h]

(a)

(b)

Fig. 6. Implementation steps: (a) compute equation (7) and initialize LUT1; (b) compute equation (2) for P-value>0.01 to initialize LUT2.

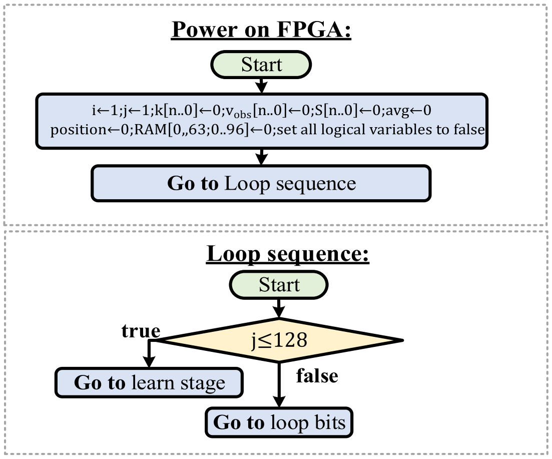

Power on FPGA:
Start
i←1;j←1;k[n..0]←0;v_obs[n..0]←0;S[n..0]←0;avg←0
position←0;RAM[0,,63;0..96]←0;set all logical variables to false
Go to Loop sequence
Loop sequence:
Start
j≤128
true
Go to learn stage
false
Go to loop bits

Fig. 7. FPGA initialization at *power up* followed by *loop sequence* for configuration testing.

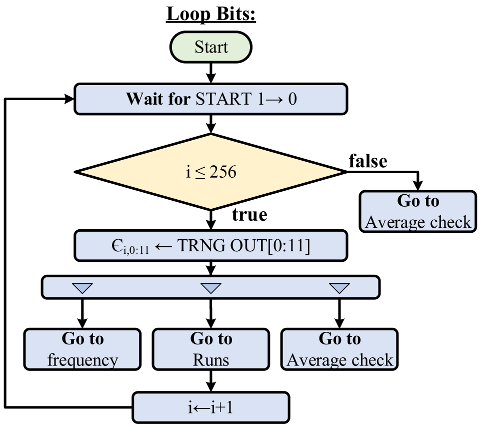


Fig. 8. Loop bits conduct the Frequency Monobit, Runs, and Maximum Average Check test.

Power-ON FPGA: The FPGA is configured to self-load using the instruction set stored in the memory. The variables are then initialized and the system now starts to execute the first of the 128 sequences to test this specific configuration as part of the Loop Sequence as shown in Fig. 7.

1. Loop sequence: The HP acquires 32768 collapses from the TRNG output and analyzes this data as 128 sequences of 256 following NIST specifications of number of sequences and sequence length for the Frequency Monobit and the Runs test. Each test result is updated after every collapse; thus, the system does not need to store the 32,768 collapses to process the data This stage checks if we have already analyzed all 128 sequences and redirect the process to learn stage for configuration storage. If all the sequences have not been analyzed, the *loop bits* procedure is initiated to execute the tests for these bits.
2. Loop bits: The falling edge of the START signal triggers the collection of 12-bit TRNG output. In this procedure shown in Fig. 8, we conduct the Frequency Monobit, the runs and the average check test, depicted in Fig. 9 and described next. Note that all tests are performed in parallel for three LSBs (index n is set to 0 to 2). The test process of the Frequency Monobit, Runs, and Max average collapse check tests is described next:
   i. *Frequency Monobit Test*: During Frequency Monobit test, the partial sum S_u is first calculated for each LSB under analysis and compared with the desired range previously determined for the sequence length u in Section III. Fig. 9(a) shows the implementation procedure for the Frequency Monobit test. Note that after 256 collapses, *pass_frequency* variable indicates whether the sequence passed (*true*) or not (*false*) the Frequency Monobit test.
   ii. *Runs Test*: This test accumulates the number of ones (k) in the sequence under analysis (ϵ_i) and calculates the number of observed runs (v_{obs}) checking if it satisfies the minimum and maximum value constraints of the *P-value* comparing to the results previously stored on LUT1. The implementation of the Runs test is shown in Fig. 9(b). As in Frequency Monobit test, after 256 collapses, *pass_runs* variable indicates whether the block passed or not the Runs test.
   iii. *Maximum Average Collapse Check*: This test accumulates the number of cycles taken for

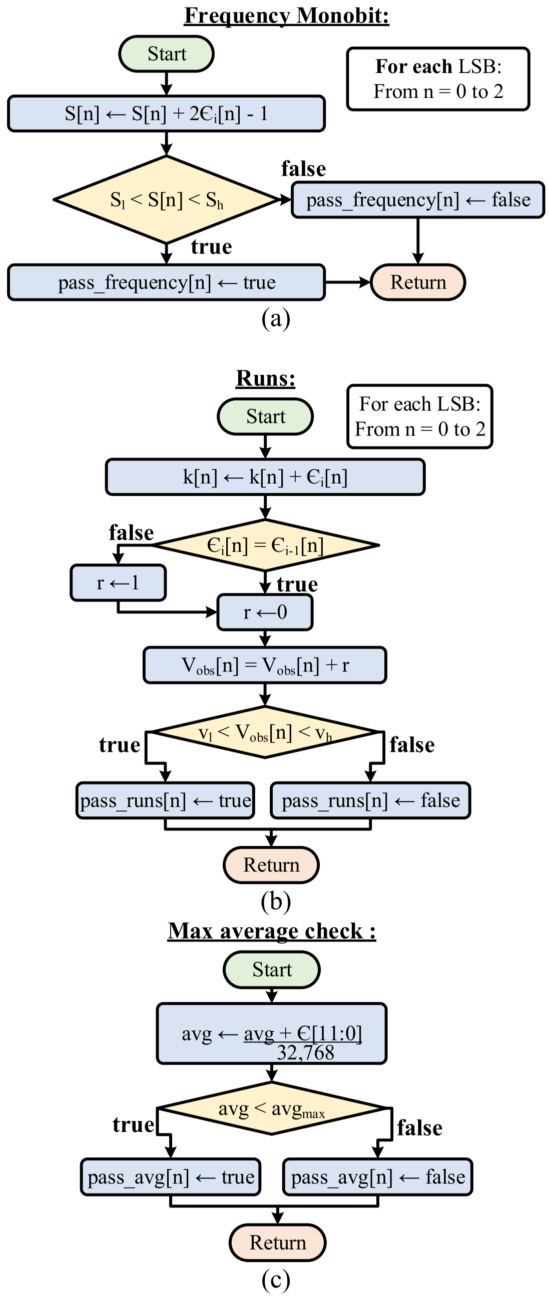


Fig. 9. Hardware implementation of (a) NIST Frequency Monobit test, (b) NIST Runs test, and (c) Maximum Average Check test.

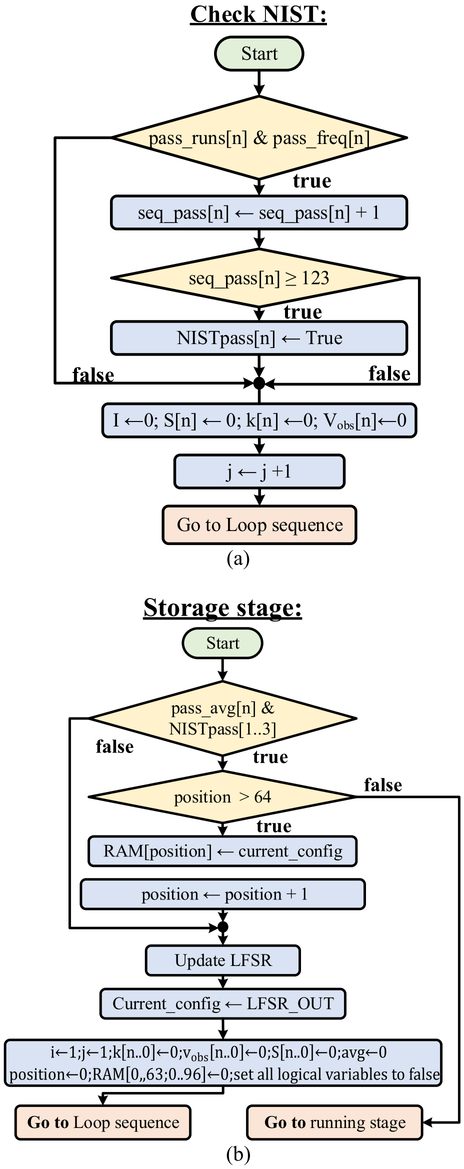

Check NIST:
Start
pass_runs[n] & pass_freq[n]
true
seq_pass[n] ← seq_pass[n] + 1
seq_pass[n] ≥ 123
true
NISTpass[n] ← True
false
false
I ←0; S[n] ← 0; k[n] ←0; V_obs[n]←0
j ← j +1
Go to Loop sequence
(a)
Storage stage:
Start
pass_avg[n] &
NISTpass[1..3]
false
true
position > 64
false
true
RAM[position] ← current_config
position ← position + 1
Update LFSR
Current_config ← LFSR_OUT
i←1;j←1;k[n..0]←0;v_obs[n..0]←0;S[n..0]←0;avg←0
position←0;RAM[0,,63;0..96]←0;set all logical variables to false
Go to Loop sequence
Go to running stage
(b)

Fig. 10. (a) Check NIST stage, and (b) storage stage.

32768 collapses and divide the outcome by 32768 to obtain the average collapse of the configuration under test, storing it in the *avg* variable. Increasing the number of cycles taken to collapse also increases the number of random channels. However, this approach can result in a bad configuration to be stored if the collapse average of a configuration sequence is very close to the maximum value. This is because the RO may not collapse when this collapse average is close to the maximum and keep oscillating. So, we must guarantee that learned configurations do not perform so close to the lock condition. The implementation is shown in Fig. 9(c) where *pass_avg* variable indicates if *avg* is below the maximum allowed value avg_{max}.

3. Check NIST: For each LSB (channel), if both variables *pass_runs* and *pass_freq* are true, a *seq_pass* variable is incremented, accumulating the number of sequences approved in the implemented NIST tests. After that, the variables are reinitialized, and a new sequence is evaluated. These steps are repeated every 128 sequences of 256 collapses. The channel under test is considered as approved in the NIST subtests if it has 123 (for 128 sequences) or more sequences approved on both Frequency Monobit and Runs tests. Each one of the 3 LSBs (channels) flags an independent flag *NISTpass*[n] if approved. The implementation procedure is shown in Fig. 10(a).
4. Storage stage: After the analysis of the 128 sequences of the configuration under test, the *storage stage* checks if all the three LSBs have been approved in NIST i.e. the variable *NISTpass* is true for all the three LSBs.

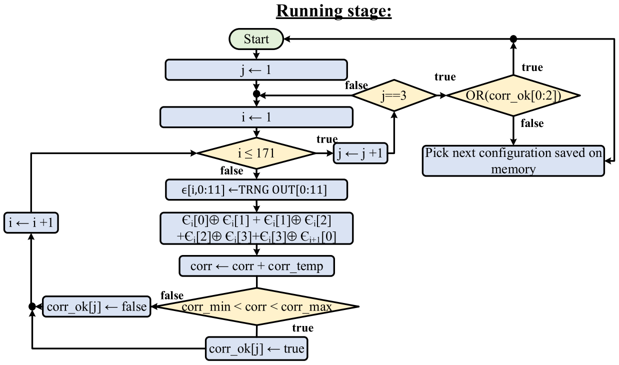


Fig. 11. Running mode algorithm.

If the calculated average of collapses is smaller than the maximum predefined then the HP will store the configuration in an internal RAM. The memory address pointer is then incremented, and the LFSR is updated for testing the next best configuration and the whole process is repeated until the desired number of stored configurations has been realized. The implementation procedure is shown in Fig. 10(b). Future work will optimize the configuration storage in an internal or external E^{2}PROM to avoid losing configurations if there are shutdown problems.

B. Running Mode

This mode represents the normal operation of the system and detects any bias variations by calculating the autocorrelation coefficient determined using 513 bits. Because the three LSBs are concatenated as a single random output the number of collapses needed to calculate a single autocorrelation coefficient are 171 ($= 513/3$). Though a single correlation coefficient can be used to predict any bias variations, we compare three consecutive autocorrelation coefficients as shown in Fig. 11. A single autocorrelation coefficient being a statistical variable will eventually fall outside the desired range; hence, the selection of three consecutive correlation coefficients ensures that false negatives are avoided. The calculation of three correlation coefficients however results in slightly higher cost requiring 513 ($=171 \times 3$) collapses in total for detection of bias variations and correction. The RO configuration is changed, and an alarm is sent (using the stored configuration) if the algorithm detects that none of the last three correlation coefficients are within the desired range. The random numbers generated are available at the output if and only if this test was approved, complying with the total failure test requirements of AIS-20/31 [19]. This test also fulfils the requirements of the NIST health test considered as a requirement for NIST compliance. Because the correlation coefficient checks the condition of the entropy source at the start up and continuously thereafter during the device operation, it detects any hardware malfunction and thus complies with the NIST requirements. Compared to TRNG in [16], the proposed tests are TRNG-architecture independent with the learning and the running modes executing different tests to achieve faster bias detection with higher probabilities of correction.

V. HP Hardware Implementation

Targeting fast data analysis and reusable HP platform, the proposed circuit implementation was implemented on two embedded boards – an Altera Cyclone V DE-10 FPGA implementing the HP and an Altera Cyclone IV FPGA implementing the TRNG. The choice of two separate boards not only provides maximum flexibility for speed and memory capacity but also allows capturing the effect of bias variations. Future research work will replace the TRNG with a faster custom integrated circuit design. The VHSIC Hardware Description Language (VHDL) was used to program the circuits. The DE10-Nano board contains a 50 MHz oscillator, hardware processor system with a dual ARM, and 1 GB DDR3 SRAM at 8 PSI. The hardware implementation of the learning mode and the running modes in the HP is shown in Fig. 12(a) and described in detail further.

The RO is implemented with two chains of 16 inverter stages, followed by a 12-bits counter to calculate the number of oscillations before the collapse and a 12-bits register used to hold the counter output. The implementation of the HP follows from the algorithmic design in Section IV and is divided into two modules called as *learning and running* respectively. These modules share common blocks such as the clock generator, bit and sequence counters, and the memory. The clock generator uses the main system clock, START, to generate four different clocks with 12.5% duty cycle labeled as Phi[3..0] as shown in Fig. 12(b). These clocks are used for pipelining the internal registers ensuring any timing conflicts are resolved. Two counters named as *cnt_bit* and *cnt_seq* track the number of bits and sequences needed to execute the Frequency Monobit and the Runs test. The *cnt_bit_cout* is set to one when 256 collapses have been analyzed. Similarly, processing 128 sequences sets the *cnt_seq_cout* to one.

A hardware processing system (HPS) is used to communicate with the ARM on the FPGA to acquire data in real-time as shown in Fig. 12(c). Although not needed for actual operation of the proposed algorithm, this data collection serves to validate the algorithm by generating some of the results in this work. The power and area for the HPS are not included in the final metrics.

A. Learning Mode Module Implementation

The implementation of learning mode can be divided into 6 main modules: i) Frequency Monobit test, ii) Runs test, iii) Maximum Average Check, iv) bit and sequence counter, v) count sequences approved, and vi) storage config decision. Note that the Frequency Monobit and Runs tests operate concurrently until the entire 256-bit sequence has been analyzed for each of the 3 LSBs.

i. The *frequency test module* increments (or decrements) the output by one if a one (or zero) is detected. The final output is then compared against the LUT2 stored range (s_l, s_h). The *pass_freq* flag is set to one if the output falls within this range. This process is repeated for each of the three LSBs.
ii. The *runs test module* is implemented using a D Flip-Flop and a comparator that compares the current TRNG

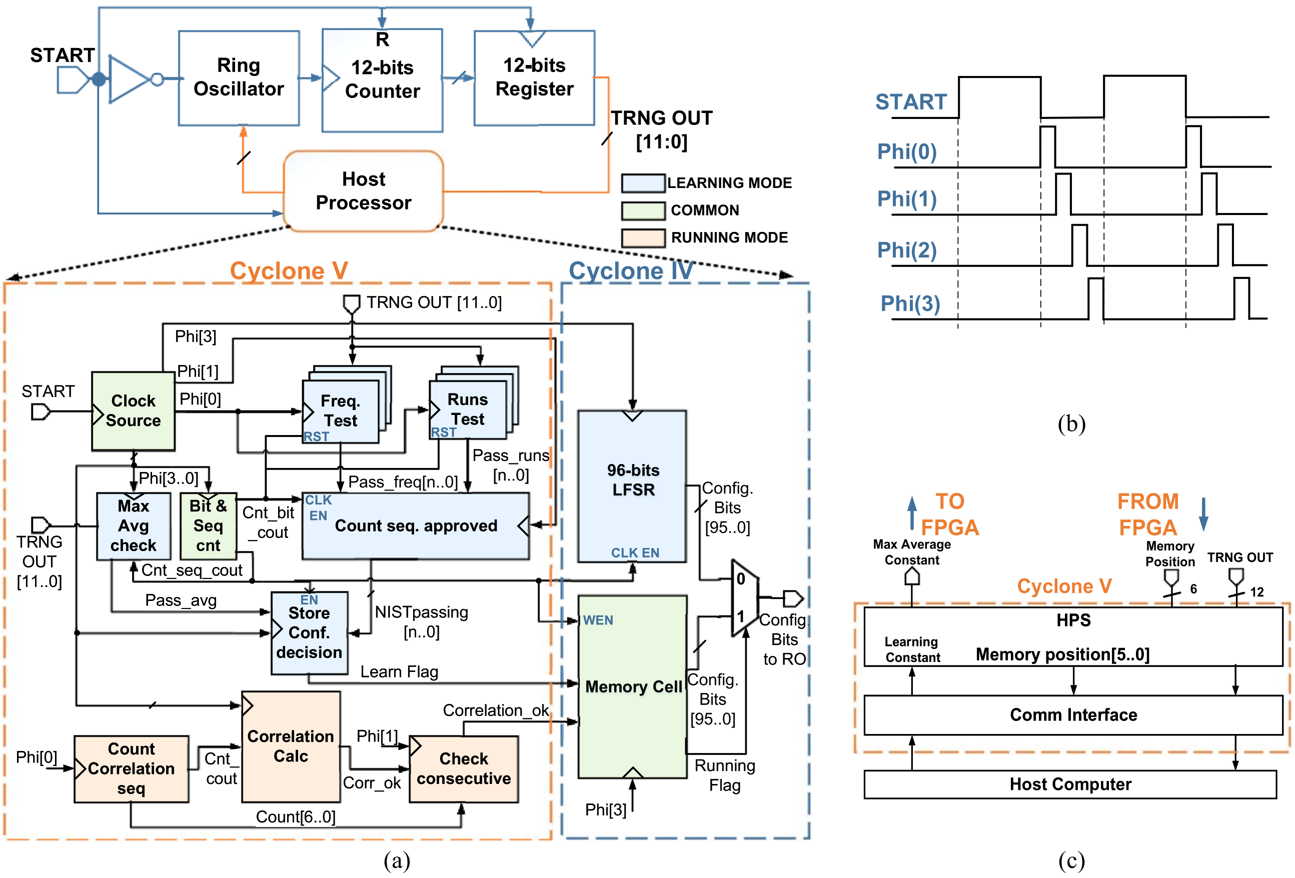


Fig. 12. (Counter-clockwise) (a) Proposed system architecture of the TRNG with the HP implemented on the FPGA, (b) timing diagram, and (c) data collection on a remote computer from the hardware processor system (HPS) on the Altera FPGA.

output with the previous cycle. A counter is incremented when the comparator bit is set high. A second counter is used to count the number of ones (k) in the sequence. The first counter output is compared with the range of values in LUT1 for the given k. The *pass_runs* flag is set to one if the counter output falls within this range. This process is also repeated for each of the three LSBs. The internal variables for both the tests are reset after the whole sequence has been analyzed. The *count sequences approved block* then reads this data to update the number of sequences and number of random channels passing the Frequency Monobit and the Runs test.

iii. *Maximum average collapse check* uses an accumulator that sums the collapse times of the 12-bit output from the TRNG. The sum times are stored in an accumulator and divided by 32,768 to obtain the average of collapses by shifting right 16 bits. The obtained value is then compared with the maximum allowed cycles as determined empirically in Section VI. If the average is found to be lower than the maximum, *pass_avg* is set to one.

iv. The *bit* and *sequence counters module* is used for both the learning and running modes. The first counter in this block counts the number of bits analyzed when the system is processing a sequence setting *cnt_bit_cout* to one each time 256 bits (one entire sequence) are analyzed. At the end of each sequence, the sequence counter is incremented by one. When all 128 sequences are analyzed, the sequence counter sets *cnt_seq_cout* flag to one.

v. The *count sequences approved* is implemented using a single counter for each channel that is incremented by one every time *pass_freq* and *pass_runs* are both one. Each one of the 3 LSB (n[2..0]) are connected to a separate counter. The counter output is compared to a constant (=123). *NISTpassing*[n] flag is set to one if the comparison is found positive.

vi. Lastly, the *storage config decision module* implements a logical AND that outputs a high level if both the inputs pass_avg and NISTpassing[n..0] are one.

After the designated memory for storing the configurations is full, and the flag *running_flag* is set. The the *frequency*, *runs* and *maximum average collapse check modules* are disabled to save power. Only the *correlation test module* is enabled leading to the start of the running mode.

B. Running Mode Module Implementation

The implementation modules in the running mode are shown in Fig. 12(a). The correlation coefficient is calculated for the random *output* vector by concatenating last 3 LSBs of TRNG. As explained in section III, at least a 512-bit sequence is needed to correctly calculate the correlation coefficient.

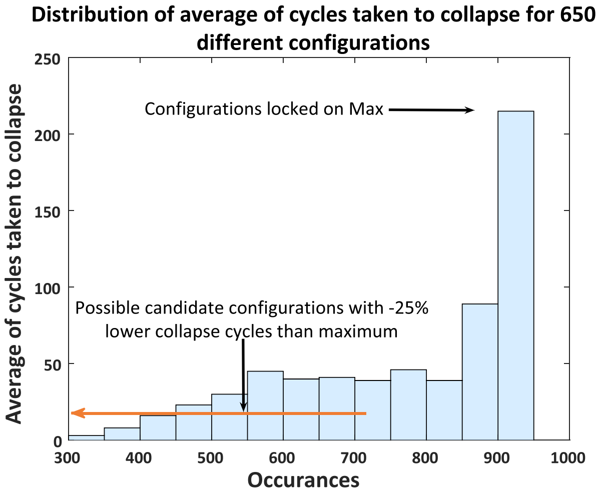


Fig. 13. Measured distribution of average of cycles to collapse for 650 configurations which will be used as possible candidates for storage.

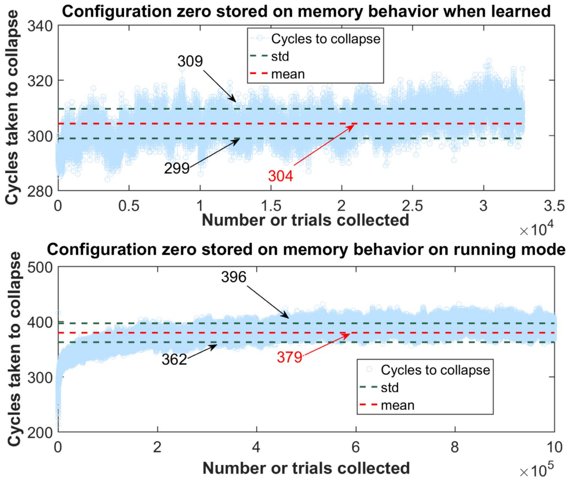


Fig. 14. Mean and standard deviation of the same stored configuration during the learning and the runing modes.

The number of collapses needed to collapse is set greater than 512 to 513 (chosen to be a multiple of 3). This thus requires 171 (= 513/3) collapses. The *count correlation seq module* counts number of collapses used. The parameter *cnt_cout* is set to one when count is equal to 171. The *correlation calc module* calculates the autocorrelation coefficient and compares with the desired range as shown in Fig. 5 for a sequence length of 513. If the calculated value is inside the desired range *corr_ok* is set to one.

The determination of biased (or un-biased) configuration is now done by ensuring that *correlation_ok* flag is one for at least one of the three consecutive sequences analyzed. If this is true, the configuration is un-biased and hence, *valid_config* flag is set to one. Otherwise, *valid_config* flag is set to zero adding the memory position vector by one. When this condition occurs, a new configuration is read from memory.

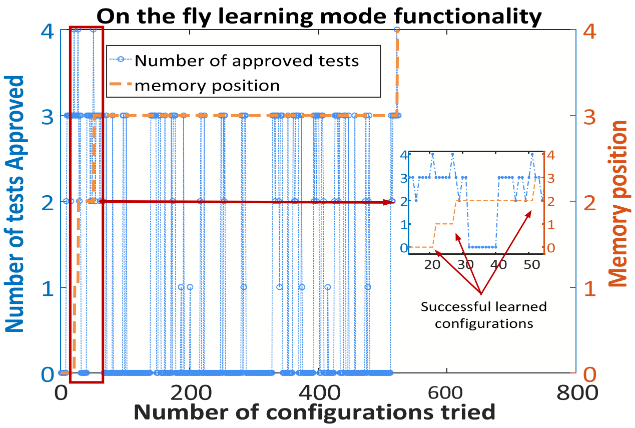


Fig. 15. Measured results showing the configurations getting stored in the memory on-the-fly after passing the Frequency Monobit and the Runs test for each of the three LSBs, and the maximum average collapse check.

VI. Measured Results

A step-by-step procedure is described for measuring the entropy during the proposed experimental testbed.

First, the max average constant (defined as $avg_{\max}$ in Fig. 9(c)) is empirically initialized before the start of the real operation (also shown in Fig. 12(c)). This constant enables the designer to account for any environmental variations (such as different oscillation frequencies) when the same hardware is implemented on different FPGA boards. The max average constant is thus defined by calculating the average of cycles that each configuration took to collapse. A total of 21 million collapses, thus 650 different configurations, were collected as shown in Fig. 13. It is evident that the maximum cycles taken by some configuration can be as high as 950 for our system. It also indicates that a configuration that does not collapse will lead to 950 oscillations at the TRNG output.

Second, the difference in core junction temperatures and its effect on the collapse average during the learning and the running modes needs to be accounted during the selection and storage of configuration in the learning mode itself. For example, as shown in Fig. 14, the same configuration stored during the learning mode will present a slightly higher average of collapse (mean) when reused during the running mode. In other words, there is a higher probability of configurations being selected and stored in memory for which a collapse hasn't occurred. This variation is found to be 25% and captured in Fig. 14. The above two tests enable the max average constant to be optimized for the learning mode implemented on different FPGA boards. Based on previous results our learning max average constant was defined as 75% × 950 = 712.

Third, the operation of the learning mode is validated by collecting the memory position pointer (defined as *position* in Fig. 10(b)) and the following output flags: *NISTpass*[2-0] and *pass_avg* as shown in Fig. 12. The configuration is saved if all these flags are one which indicates that all the 3 LSBs passed the Frequency Monobit and the Runs tests and the calculated maximum average is lower than the max constant defined earlier. Fig. 15 shows the configuration selection and storage in real-time. The memory position is incremented from 1 to 4 as four configurations passes the above tests as shown in the inset in Fig. 15.

Fourth, after the successful selection and storage of configurations, the proposed method using stored configuration

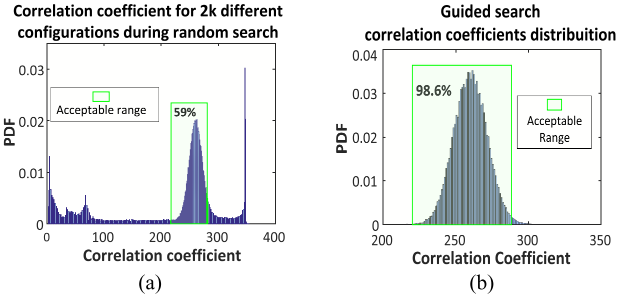


Fig. 16. Measured results showing comparison of the proposed guided search technique to a random search yielding higher success with significantly lower number of collapses.

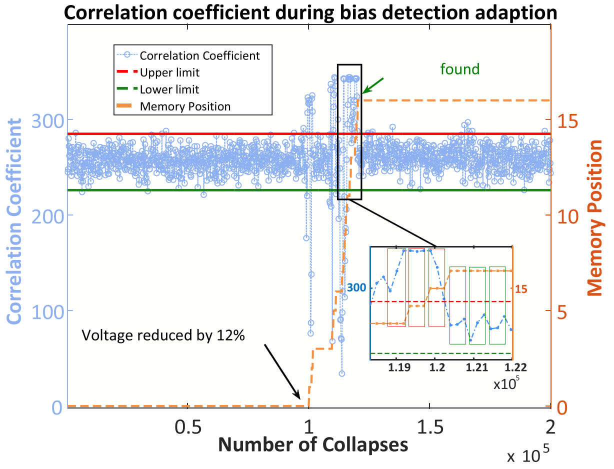


Fig. 17. System adaption in real-time when the supply voltage is reduced from 1.25 V to 1.1V (around 12% change).

(from the learning mode) is compared against prior art approach [16] using brute-force random search. The first search uses randomly generated configuration bits to identify a good configuration. The second search uses the configuration bits learned using the proposed method. Fig. 16 shows the results of this comparison confirming the latency improvements of the proposed method over prior art. The use of autocorrelation coefficient instead of average of collapses (as a metric earlier adopted in [16]) yields higher success rates with a significantly smaller number of collapses to validate a configuration. Hence, the detection of any bias variations and subsequent correction will need much less clock cycles being much faster.

Fifth, the effect of bias variations is evaluated by introducing intentional supply variations. Figs. 17 and 18 shows two experiments where the supply voltage is reduced by 12% and 33% respectively and the proposed method detects the bias variations and find new stable configurations on-the-fly successfully, where the entropy of the output became high again. Fig. 18 also shows that the system does not stay stable forever but will have stable moments where we can gather random output, and it will always adapt to a more stable configuration where high entropy bitstreams can be extracted where they are stable. The above results are further confirmed by reading the data off-line to a computer and evaluated against a complete NIST test suite as shown in Table II.

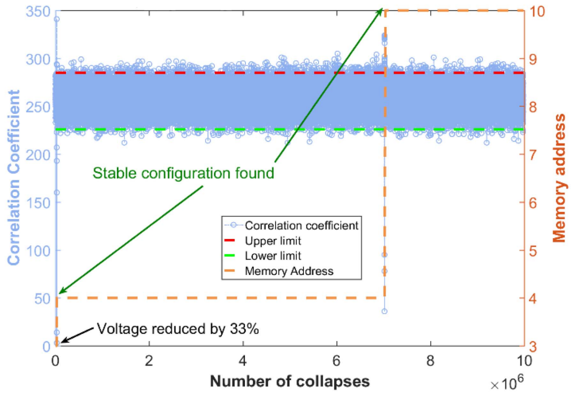
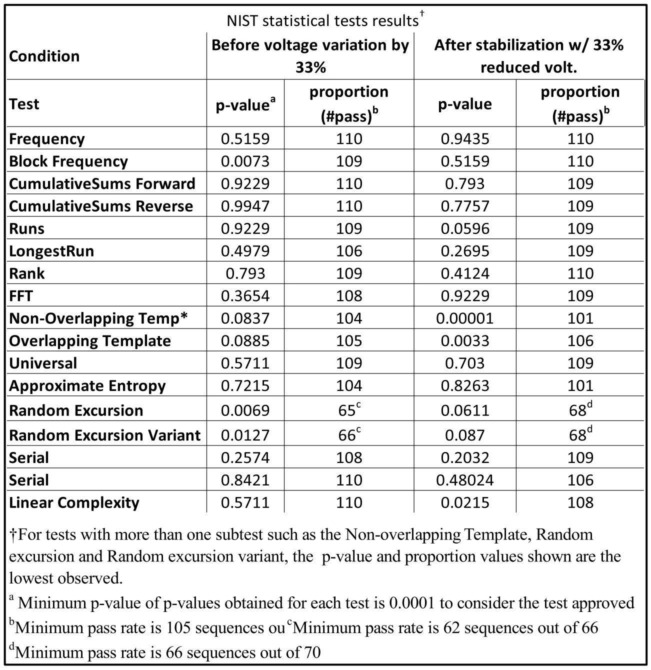


Fig. 18. Real-time system reconfiguration when a supply voltage change of 33% (1.2 V to 0.8 V) is detected.

TABLE II
NIST STATISTICAL TEST RESULTS



Condition	NIST statistical tests results† Before voltage variation by 33%		After stabilization w/ 33% reduced volt.	
Test	p-value[a]	proportion (#pass)[b]	p-value	proportion (#pass)[b]
Frequency	0.5159	110	0.9435	110
Block Frequency	0.0073	109	0.5159	110
CumulativeSums Forward	0.9229	110	0.793	109
CumulativeSums Reverse	0.9947	110	0.7757	109
Runs	0.9229	109	0.0596	109
LongestRun	0.4979	106	0.2695	109
Rank	0.793	109	0.4124	110
FFT	0.3654	108	0.9229	109
Non-Overlapping Temp*	0.0837	104	0.00001	101
Overlapping Template	0.0885	105	0.0033	106
Universal	0.5711	109	0.703	109
Approximate Entropy	0.7215	104	0.8263	101
Random Excursion	0.0069	65[c]	0.0611	68[d]
Random Excursion Variant	0.0127	66[c]	0.087	68[d]
Serial	0.2574	108	0.2032	109
Serial	0.8421	110	0.48024	106
Linear Complexity	0.5711	110	0.0215	108

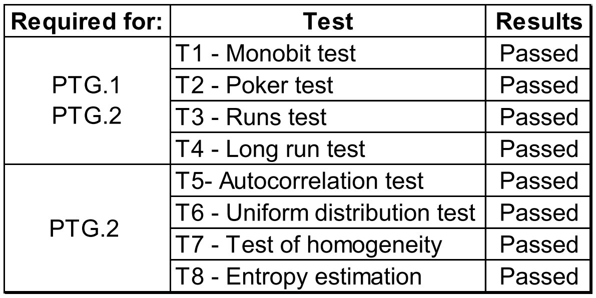
†For tests with more than one subtest such as the Non-overlapping Template, Random excursion and Random excursion variant, the p-value and proportion values shown are the lowest observed.

[a] Minimum p-value of p-values obtained for each test is 0.0001 to consider the test approved

[b]Minimum pass rate is 105 sequences ou [c]Minimum pass rate is 62 sequences out of 66

[d]Minimum pass rate is 66 sequences out of 70

TABLE III
AIS-20/31 ONLINE TEST RESULTS



Required for:	Test	Results
PTG.1 PTG.2	T1 - Monobit test	Passed
	T2 - Poker test	Passed
	T3 - Runs test	Passed
	T4 - Long run test	Passed
PTG.2	T5- Autocorrelation test	Passed
	T6 - Uniform distribution test	Passed
	T7 - Test of homogeneity	Passed
	T8 - Entropy estimation	Passed

All recent TRNGs for cryptographic applications must comply with both the AIS-20/31 and the NIST recommendations. Hence the same data was evaluated using the AIS-20/31 tests proposed by the German Federal Office for Information in Security's (BSI) for TRNG classification [19].

TABLE IV

COMPARISON OF THE PROPOSED EMBEDDED ON-THE-FLY TRNG / HP ALGORITHM WITH STATE-OF-THE-ART

		This work		[16] JSSC'16	[26] CICC'14	[27] TCAS-II'17
Randomness source		**Reconfigurable RO**		Reconfigurable RO	Beat frequency of free-running RO	Beat frequency of two free-running RO
Technology		**Altera Cyclone V DE-10**	**Altera Cyclone IV**	40nm CMOS	Xilinx Virtex V	Xilinx Virtex V
Block		**Host processor (HP)**	**Reconfigurable TRNG**	Reconfigurable TRNG (HP off-chip)	TRNG	TRNG
Area[g]	Dedicated Registers	**179[a]**	**229[a]**	$836\ \mu m^2$	26[f]	26[f]
	LUTs	**295[a]**	**343[a]**		48[f]	33[f]
	Memory	**N/A**	**3072**		N/A	N/R
	Clock Manager	**N/A**	**N/A**		1[b]	2[c]
Power (mW)		**10.96**	**10.22**	0.046	1384	1470
Output frequency (kHz)		**6**		667	N/R	223
Number of random outputs		**3**		3	4	3
Recovery latency (#clock cycles)		**522**		6250-62500[e]	No on-the-fly correction	No on-the-fly correction
success probability		**98% (guided search)** **59% (random search)**		5-8%		
Post Processor		**No**		No	Yes	Yes

[a]Area not optimized; [b]Phase-locked loop; [c]Digital clock manager; [d]512 trials/ probability of strong configuration (=0.98);
[e](500-5000 trials) / 0.08 (reported success rate for strong configuration); [f]No embedded host processor for PVT control and reconfigurable TRNG. The hardware footprint excludes the MicroBlaze soft processor (necessary for overall control and data acquisition from the TRNG), and 46 bytes of memory.
[g]Link to compare resources usage between Xilinx and Altera: https://www.altera.com/en_US/pdfs/literature/wp/wp-01040.pdf
N/A – Not applicable; N/R – not reported;

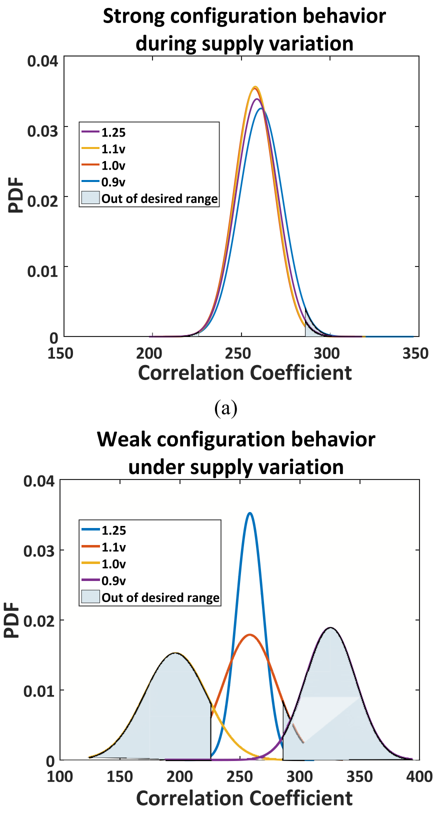
The BSI methodology recommends that the physical TRNGs fulfill the requirements of PTG.2 class [19]. The class PTG.2 requires that the RNG passes a total failure test that detects a total failure of entropy source when the RNG has started. If detected the TRNG should not output any random number. Also, if a total failure occurs while the device is being operated the same test must prevent that any output is passed on to TRNG dependent devices. This requirement was fulfilled by the proposed topology in this work. At the beginning of the Running mode, the autocorrelation coefficient is also used as a total failure test. The output is thus buffered if and only if the analyzed sequence was approved.

Further, PTG.2 requires that an online test is applied on the raw random sequence (in absence of any post-processing) both during the start and normal operation of the TRNG. The online tests detect non-tolerable statistical defects of the internal random numbers. Because the application developed here is not integrated in a large system where the online test could be called, we have performed all the tests offline using BSI's test suite [19]. The same data used in the NIST test suite was evaluated and passed all online tests (T1-T8) required for an AIS-20/31 compliant device aiming PTG.2 certification. Table III shows the obtained results. Finally, the PTG.2 certificate requires that the average Shannon entropy per internal random bit exceeds 0.997. The proposed TRNG obtained a Shannon entropy of 0.999 after bias correction and stabilization (from the data in Fig. 17).

Finally, we observe the behavior of the correlation coefficient to prove that the proposed system is highly adaptable. As shown in Fig. 19(a). the proposed approach works because some configurations keep producing high entropy bits even though the collapse average has shifted due to process, temperature or voltage variations as their correlation coefficient is in the desired range, differently of what was proposed in [16] where a narrow average shift was the metric for bias detection.

The other configurations (shown in Fig.18(b).) however may lock to the maximum collapse value or decrease it oscillation average to a much smaller value unable to produce the three random channels. Their output will thus be biased causing the correlation coefficient to shift outside the desired range.

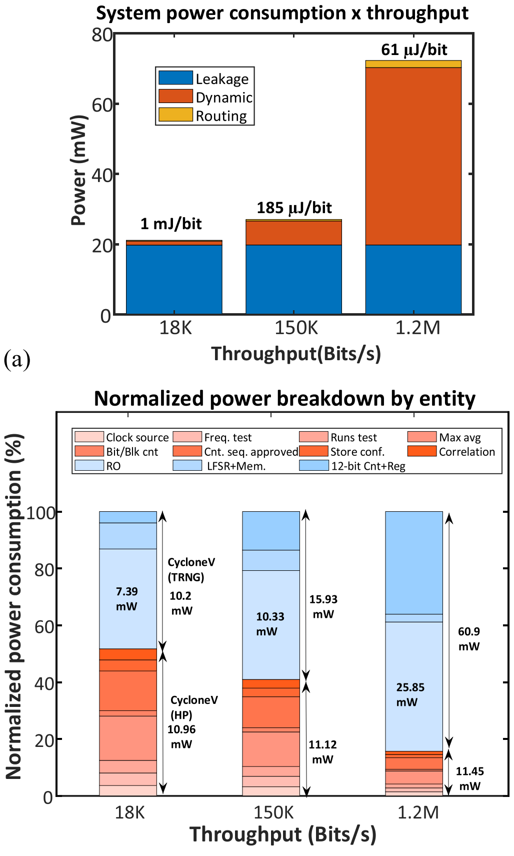
Table IV compares the proposed work to state-of-the-art. The measured power consumption of the TRNG and the HP is 10.96mW and 10.22mW respectively at 1.25V supply at a throughput of 18 kHz for three random channels. As higher throughputs are limited by the internal delays of the RO currently implemented using Cyclone V FPGA, we used Altera Powerplay Power Analyzer [28] and Modelsim-Intel [29] software tools to estimate the scalability and power consumption of the proposed architecture for different simulated throughputs.



(a)

(b)

Fig. 19. Response to the supply variations with (a) strong, and (b) weak configurations.



(a)

(b)

Fig. 20. (a) System power consumption for different throughputs classified by leakage, dynamic and routing power breakdown, and (b) normalized power breakdown by entity in percentage for three throughputs.

Fig. 20(a) shows that the proposed architecture is scalable with the dominant power consumption due to the leakage power. The energy efficiency of the proposed architecture at 18kB/s throughput is 1mJ/bit. Fig. 20(b) presents the normalized power breakdown by entity with the 32-stage RO being the main power constraint. The RO power consumption can be improved using an ASIC implementation as in [16]. Given that the focus of the proposed work is the bias detection and correction, it can be observed that there is only 4.5% increase in power consumption when the simulated throughput is scaled more than 600×. The area and the power consumption will be further optimized in further works using an ASIC implementation.

Further the actual processing latency will vary substantially between an ASIC and a FPGA architecture that makes it extremely difficult to compare. The choice for FPGA implementations over ASIC comes with tradeoffs (consumption versus flexibility) [30]. Hence, Table IV uses the number of cycles as a metric because it is technology independent. Also, it is important to note that none of the other FPGA implementations in Table IV use any bias detection and correction mechanism, and recovery latency is up to 120 times faster when compared with [16], which are the main contributions of this work.

VII. Conclusions And Future Works

This work demonstrates on-the-fly bias detection and correction using a reconfigurable TRNG with significant improvement in probability of bias correction over prior art and at low-latency. This is accomplished by a lightweight test suite that implements on an FPGA a subset of NIST tests for learning and autocorrelator function for bias detection and correction with large bias variations. The proposed algorithmic steps further account for supply and temperature variations on-the-fly and is highly portable to other digital TRNG architectures.

The presented architecture is highly scalable and thus extensible to several directions in future including autonomous sensor networks, sparse signal processors and IoT devices. The selection and storage of configuration bits in memory during the learning mode can be extended to encompass different environmental behaviour. In addition, a memory sorting algorithm can be designed that leverages additional information on PVT variations from the on-chip sensors to make an integrated system. The experimental testbed can be further scaled to increase the number of random channels and apply towards key authentication and reconfiguration for edge computing applications.

References

[1] D. Evans. *The Internet of Things: How the Next Evolution of the Internet is Changing Everything*. Accessed: Sep. 13, 2018. [Online]. Available: https://www.cisco.com

[2] M. Rostami, F. Koushanfar, and R. Karri, “A primer on hardware security: Models, methods, and metrics,” *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[3] H. Kumarage, I. Khalil, A. Alabdulatif, Z. Tari, and X. Yi, “Secure data analytics for cloud-integrated Internet of Things applications,” *IEEE Cloud Comput.*, vol. 3, no. 2, pp. 46–56, Mar. 2016.

[4] C. O’Flynn and Z. D. Chen, “ChipWhisperer: An open-source platform for hardware embedded security research,” in *Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

[5] S. K. Mathew *et al.*, “μRNG: A 300-950mV, 323Gbps/W all-digital full-entropy true random number generator in 14nm FinFET CMOS,” *IEEE J. Solid-State Circuits*, vol. 51, no. 7, pp. 1695–1704, Jul. 2016.

[6] N. Liu, N. Pinckney, S. Hanson, D. Sylvester, and D. Blaauw, “A True Random Number Generator using time-dependent dielectric breakdown,” in *Symp. VLSI Circuits—Digest Tech. Papers*, Jun. 2011, pp. 216–217.

[7] C. Tokunaga, D. Blaauw, and T. Mudge, “True random number generator with a metastability-based quality control,” *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 78–85, Jan. 2008.

[8] B. Sunar, W. J. Martin, and D. R. Stinson, “A provably secure true random number generator with built-in tolerance to active attacks,” *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 109–119, Jan. 2007.

[9] Y. Liu, R. C. C. Cheung, and H. Wong, “A bias-bounded digital true random number generator architecture,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 1, pp. 133–144, Jan. 2017.

[10] G. Taylor and G. Cox, “Behind Intel’s new random-number generator,” in *Proc. IEEE Spectrum*, Aug. 2011.

[11] J. D. J. Golic, “New methods for digital generation and postprocessing of random data,” *IEEE Trans. Comput.*, vol. 55, no. 10, pp. 1217–1229, Oct. 2006.

[12] A. Cherkaoui *et al.*, “A very high speed true random number generator with entropy assessment,” in *Proc. Int. Workshop Crytograph. Hardw. Embedded Syst. (CHES)*, in Lecture Notes in Computer Science, vol. 8086. Springer, 2013, pp. 179–196.

[13] V. Fischer and M. Drutarovský, “True random number generator embedded in reconfigurable hardware,” in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, in Lecture Notes in Computer Science, vol. 2523, Santa Barbara, CA, USA, Springer, 2002, pp. 415–430.

[14] M. Epstein *et al.*, “Design and implementation of a true random number generator based on digital circuit artifacts,” in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, in Lecture Notes in Computer Science. Berlin, Germany: Springer, 2003, pp. 152–165.

[15] T. Amaki, M. Hashimoto, and T. Onoye, “A process and temperature tolerant oscillator-based True Random Number Generator with dynamic 0/1 bias correction,” in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2013, pp. 133–136.

[16] K. Yang, D. Blaauw, and D. Sylvester, “An all-digital edge racing true random number generator robust against PVT variations,” *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 1022–1031, Apr. 2016.

[17] M. Dichtl, “Bad and good ways of post-processing biased physical random numbers,” in *Proc. Int. Workshop Fast Softw. Encryption*. Berlin, Germany: Springer, 2007, pp. 137–152.

[18] E. Barker and J. Kelsey. (Nov. 07, 2016). *NIST Draft Special Publication 800-90C*. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90c.pdf

[19] W. Killmann and W. Schindler, “A proposal for: Functionality classes for random number generators,” Ph.D. dissertation, Bundesamt für Sicherheit der Informationstechnik, Berlin, Germany, 2011.

[20] A. A. Abidi, “Phase noise and jitter in CMOS ring oscillators,” *IEEE J. Solid-State Circuits*, vol. 41, no. 8, pp. 1803–1816, Aug. 2006.

[21] *Security Requirements for Cryptographic Modules*. Standards FIPS 140-2, Federal Information Processing Standards, 2001.

[22] V. B. Suresh, D. Antonioli, and W. P. Burleson, “On-chip lightweight implementation of reduced NIST randomness test suite,” in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 93–98.

[23] F. Veljković, V. Rožić, and I. Verbauwhede, “Low-cost implementations of on-the-fly tests for random number generators,” in *Proc. Design Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 959–964.

[24] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, “TOTAL: TRNG on-the-fly testing for attack detection using lightweight hardware,” *IEEE Design, Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2016, pp. 127–132.

[25] M. Abramowitz and I. A. Stegun, Eds. “Repeated Integrals of the Error Function,” in *7.2 Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*, 9th ed. New York, NY, USA: Dover, 1972, pp. 299–300.

[26] Q. Tang, B. Kim, Y. Lao, K. K. Parhi, and C. H. Kim, “True random number generator circuits based on single- and multi-phase beat frequency detection,” in *Proc IEEE Custom Integr. Circuits Conf.*, Sep. 2014, pp. 1–4.

[27] A. P. Johnson, R. S. Chakraborty, and D. Mukhopadyay, “An improved DCM-based tunable true random number generator for Xilinx FPGA,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 4, pp. 452–456, Apr. 2017.

[28] Powerplay Analyzer. *Intel Quartus Prime Standard Edition User Guide: Power Analysis and Optimization*. Accessed: Sep. 28, 2019. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/xhv1529966780595.html

[29] Modelsim-Intel. *Intel FPGA Simulation—ModelSim–Intel FPGA*. Accessed: Sep. 28, 2019. [Online]. Available: https://www.intel.com/content/www/br/pt/software/programmable/quartus-prime/model-sim.html

[30] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

Leonardo Bosco Carreira received the B.Sc. degree in electrical engineering from the University of São Paulo, São Carlos, Brazil, in 2019. He is currently pursuing the M.S. degree in electrical engineering with Washington State University, Pullman, WA, USA. His current research interests include integrated hardware security techniques using random number generators for IoT systems, mixed-signal IC design, and robust adaptive control.

Paige Danielson received the B.S. degree in electrical engineering from Washington State University, Pullman, WA, USA, in 2019. She is currently pursuing the Ph.D. degree in electrical engineering with the University of Colorado at Boulder with a focus in electromagnetics, RF, and microwaves. She worked in the Systems on Chips Lab, Washington State University, from 2018 to 2019. Her current research interests are wireless communication and RF, and millimeter wave front end systems.

Arya A. Rahimi received the B.S. and M.S. degrees from Washington State University in 2013 and 2019, respectively. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science, Washington State University, Pullman. His research is focused on Wireless Body Sensor Networks (WBSNs), bio-signal acquisition, ultralow power analog signal processing, and adaptive sampling architectures using classification algorithms.

Maximiliam Luppe received the B.S. degree in computational physics and the M.S. and Ph.D. degrees from the University of São Paulo, São Carlos, Brazil, in 1994, 1997, and 2003, respectively. He is currently a Doctor Professor with the Electrical and Computer Engineering Department, Engineering School of São Carlos, University of São Paulo. His research interests include ASIC and FPGA digital design implementations, architectures for low level image processing, and the application of image processing in precision agriculture.

Subhanshu Gupta (S’03–M’11–SM’16) received the B.E. degree from the National Institute of Technology (NIT), Trichy, India, in 2002, and the M.S. and Ph.D. degrees from the University of Washington in 2006 and 2010, respectively. From 2011 to 2014, he was with the RFIC Group, Maxlinear, Inc., where he worked on silicon transceivers and data converters. He is currently an Assistant Professor of electrical engineering and computer science with Washington State University. His research interests include ultra-low-power circuits and systems, wideband wireless transceivers, and stochastic hardware optimization techniques.

He was a recipient of the Analog Devices Outstanding Student Designer Award in 2008, the IEEE RFIC Symposium Best Student Paper Award (3^{rd} place) in 2011, and the National Science Foundation CAREER Award in 2020. He served as a Guest Editor for the IEEE TRANSACTIONS OF CIRCUITS AND SYSTEMS—I: REGULAR PAPERS and the *IEEE Design and Test Magazine* in 2019. He will serve as the Associate Editor for the IEEE TRANSACTIONS OF CIRCUITS AND SYSTEMS—I: REGULAR PAPERS from 2020 to 2021.